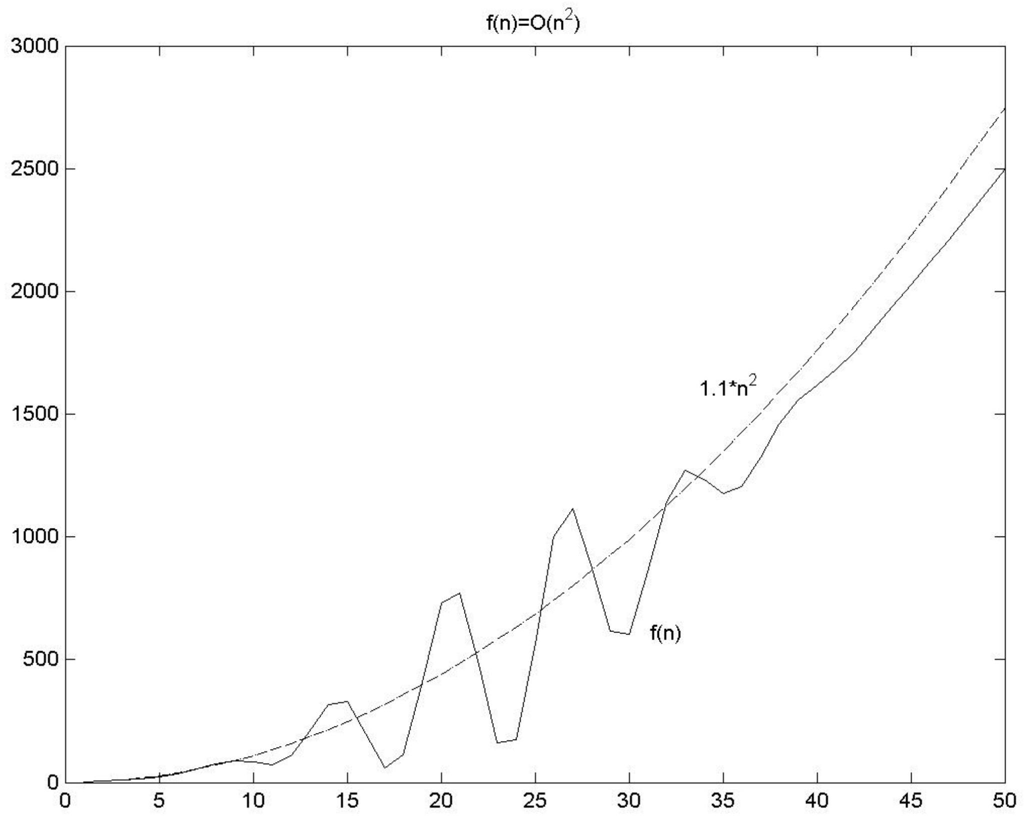
Complexity Analysis: Theta Notation

Complexity Analysis is one of the more complicated topics in the subject Mathematics For Computing. It involves an unusual concept and some tricky algebra. In these notes we try to demystify the idea a little.

Theta notation relates to analysing an algorithm and finding the number of operations it performs. At this stage we won't specify what operation we are counting. It could be additions, multiplications, matrix multiplications, recursive calls, etc. All we know is that there is some operation that we count, and the number of operations performed will depend on the size of the problem, *n*. The measure of the size of the problem, *n*, may itself be complicated. For example *n* might be the number of items to be sorted, or the size of a matrix. For now, let us just assume that *n* measures the size of the problem, and that *f*(*n*) is the number of operations needed to solve the problem when the input size is *n*. The faster *f*(*n*) grows, the slower the algorithm will be, because the number of operations, ie the amount of work, performed by the algorithm is growing rapidly.

We try to gain an understanding of how fast *f*(*n*) grows with *n.* In fact what we are seeking, technically speaking is upper and lower bounds, so that we can say "*f*(*n*) grows no faster than *x*" and "*f*(*n*) grows faster than *y*". If possible we try to get *x* and *y* to differ by no more than a constant multiple, for then we have considerable knowledge about how fast *f*(*n*) grows.

The formal definition of theta notation is a two part definition. We seek first of all an upper bound, some function *g*(*n*), say. We then say that *f*(*n*) is of order at most *g*(*n*), or $f(n) = \mathrm{O}(g(n))$. Technically, if we can find constants c_1 and n_1 with $|f(n)| \le c_1 |g(n)|$ for all $n \ge n_1$, then we say $f(n) = \mathrm{O}(g(n))$. The situation this describes is illustrated below in Figure 1.



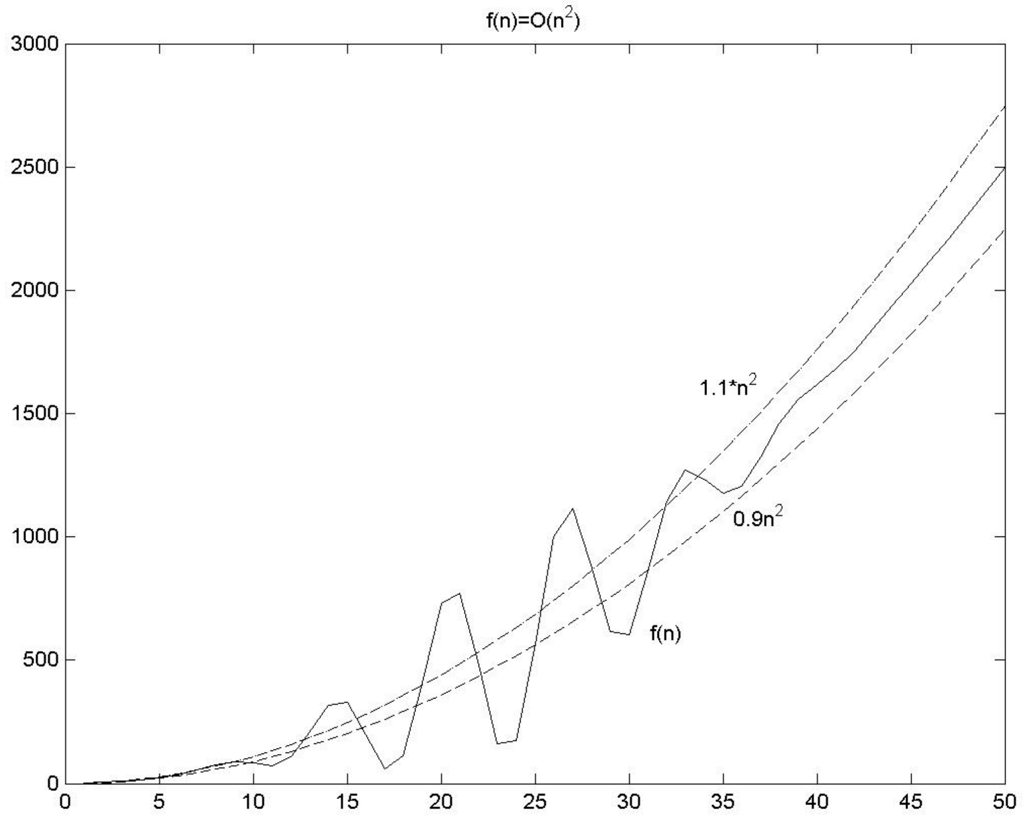
Note that *f*(*n*) is allowed to jump around and be larger than *g*(*n*) early on, but must eventually reach a point where it remains lower than the curve $c_1 g(n)$. In this case $g(n) = n^2$, and after a certain point, about *n*=35, we can say that $f(n) \le 1.1 g(n)$, so we say $f(n) = O(n^2)$[1]. Students always wonder where the constant, in this case 1.1, went to. The fact is that the value of c_1 is unimportant. We only need to know that such a c_1 exists, its actual value is unimportant. However, I find it helpful to students to rephrase the statement as follows: $f(n) = O(n^2)$ with c_1=1.1. Students tend to be baffled at the way they have to work so hard to find c_1 and then don't actually use it. So I like to use it in the statement, even though technically it is sadly irrelevant.

Now we seek a lower bound for *f*(*n*). Precisely, we seek to find constants c_2 and n_2 so that $|f(n)| \ge c_2 |g(n)|, n \ge n_2$. If we can do this, we say that *f*(*n*) is of order at least *g*(*n*), and we write $f(n) = \Omega(g(n))$. If we can find the constants c_1 and c_2, so that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we say *f*(*n*) is of order *g*(*n*), and we write $f(n) = \Theta(g(n))$. This means we know quite a lot about the growth of *f*(*n*), and is the ideal situation.

In this case it turns out that the same function, n^2, this time multiplied by 0.9, will pass under the function *f*(*n*), apart from a few glitches early on. The diagram below shows that *f*(*n*) falls below $0.9n^2$ a few times, but after that

[1] Notice that because *f* and *g* are always positive, the modulus signs are irrelevant.

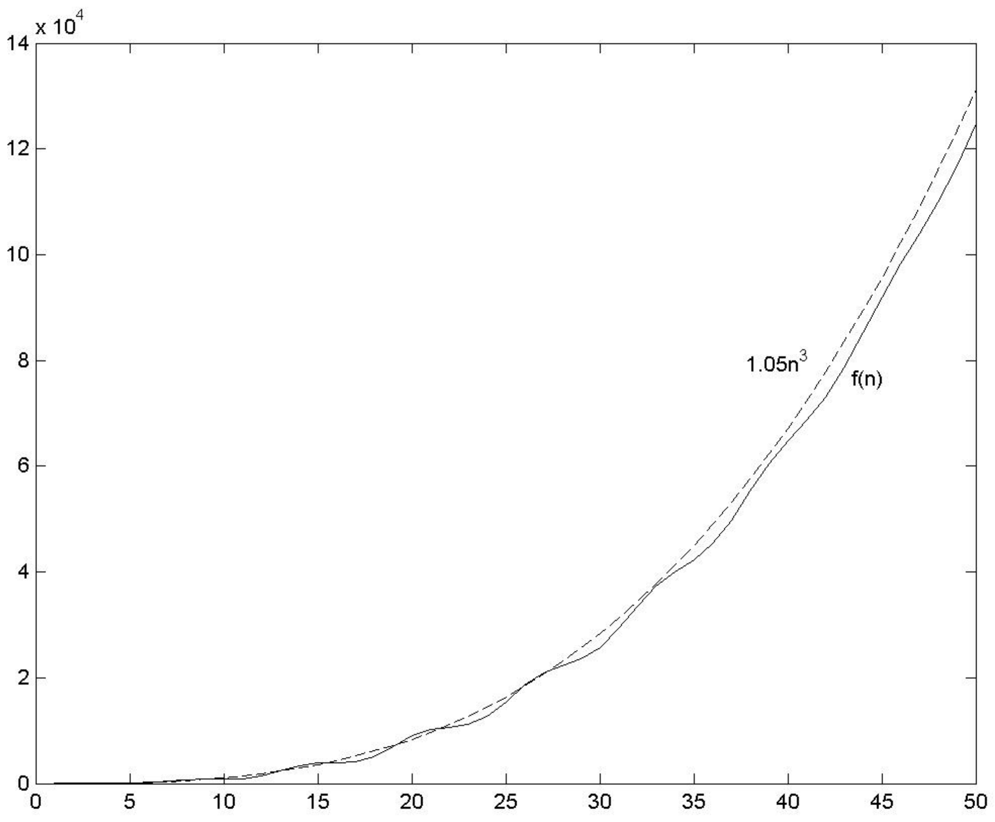
stays above the curve[2]. Hence we can say $f(n) = \Omega\left(n^2\right)$, and so $f(n) = \Theta(n^2)$.



Now try the following multiple choice questions, each related to a single diagram. In each case choose the statement that best fits the diagram.

1.

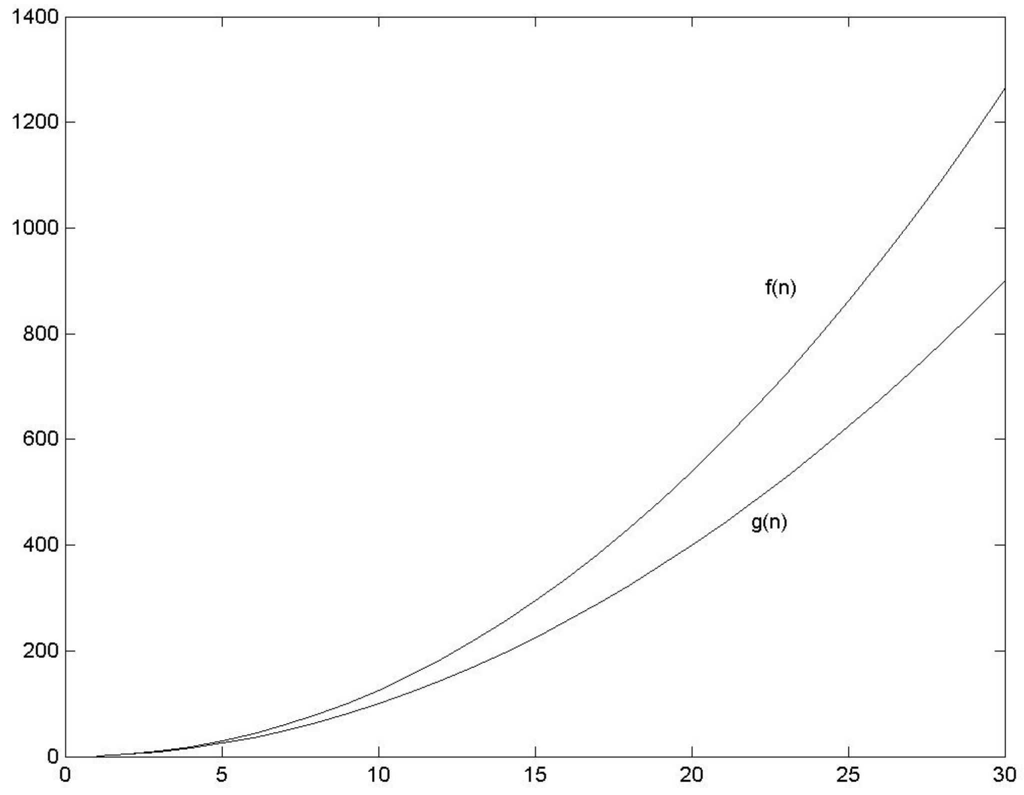
[2] Technically speaking, of course, a diagram like this can't *prove* that $f(n)$ will always remain above or below $c_1g(n)$ or $c_2g(n)$. That requires some algebra. However I have implicitly assumed that the function will always remain between the two dashed curves.



(A) $f(n) = \Omega(n^3)$ (B) $f(n) = \mathrm{O}(n^3)$ with constant 1.05

(C) $n^3 = \mathrm{O}\big(f(n)\big)$ (D) $f(n) = \Theta(n^3)$

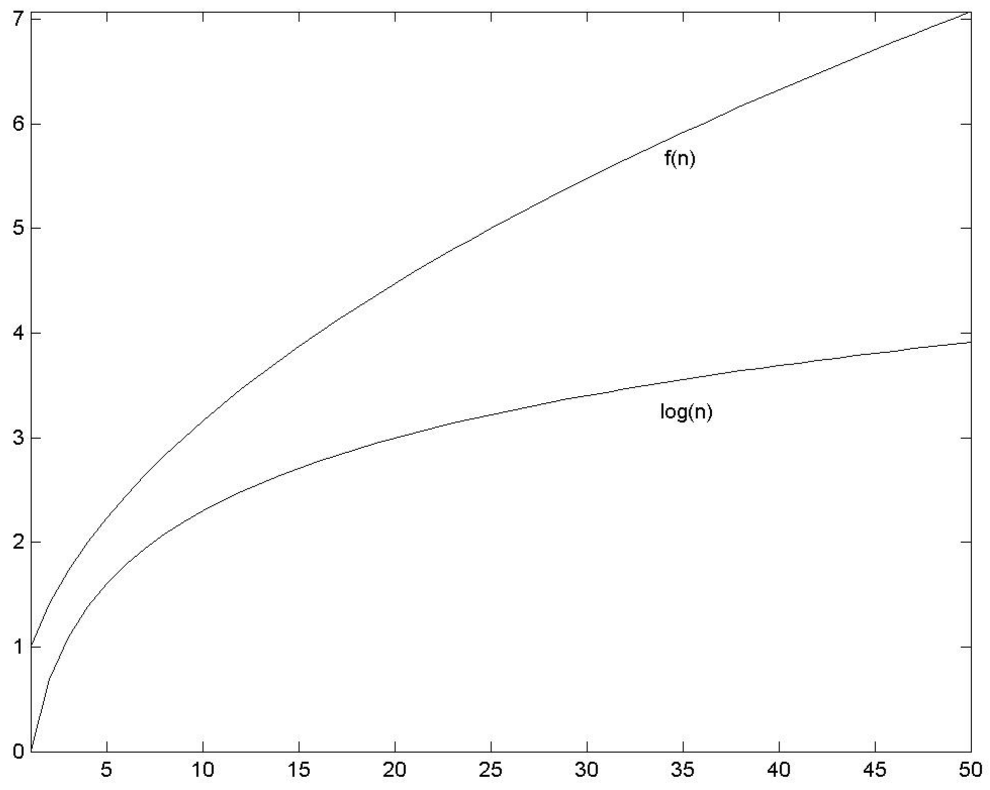
2.



(A) $g(n) = \Theta\left(f(n)\right)$ (B) $g(n) = \Omega\left(f(n)\right)$

(C) $f(n) = O\left(g(n)\right)$ (D) $f(n) = \Omega\left(g(n)\right)$

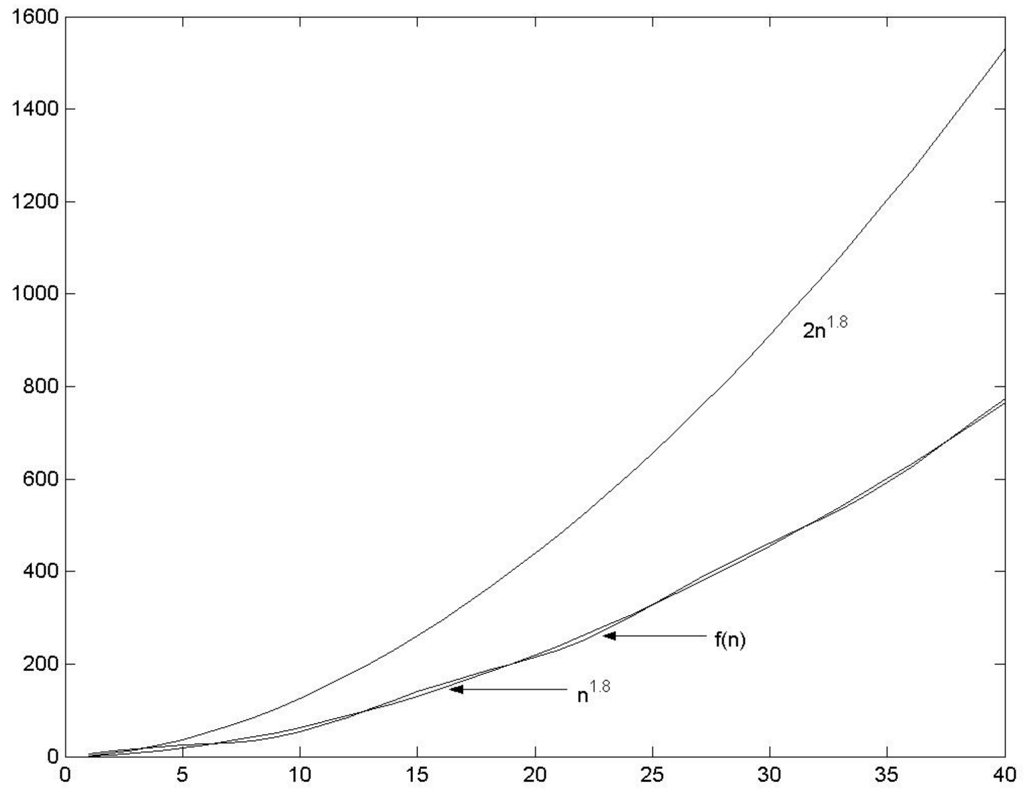
3.



(A) $f(n) = \Omega(\log(n))$ (B) $\log(n) = \Omega(f(n))$

(C) $\log(n) = \Theta(f(n))$ (D) $f(n) = O(\log(n))$

4.



(A) $n^{1.8} = \Theta\left(f(n)\right)$ (B) $f(n) = \Omega\left(n^{1.8}\right)$

(C) $f(n) = \mathrm{O}\left(n^{1.8}\right)$ (with constant $c_1 = 2$) (D) $n^{1.8} = \mathrm{O}\left(f(n)\right)$ (with constant $c_1 = 2$)

Answers

1. B 2. D 3. A 4. C

Selecting the Fastest Growing Term

When we know how many operations an algorithm will take, we can easily find the theta notation for that algorithm. This is because the number of operations will be dominated by the fastest growing term. (Remember, of course, that the faster the number of operations grows, the slower will be that algorithm.) Consider the following examples, where *f*(*n*) is the hypothesised number of operations for some unknown algorithm.

Example 1

Suppose $f(n) = n^2 + 1.3n * \lg(n) + 4n + 5$[3]. The table below shows the value of *f*(*n*) and also the function $g(n) = n^2$ and the percentage difference between them, for certain values of *n*[4].

n	f(n)	g(n)	% difference
1	10.00	1	-90.000%
10	188.19	100	-46.861%
100	11268.70	10000	-11.259%
1000	1016960.52	1000000	-1.668%
10000	100212745.26	100000000	-0.212%
100000	10002559258.26	10000000000	-0.026%
1000000	1000029911044.14	1E+12	-0.003%
10000000	100000342295462.00	1E+14	0.000%
100000000	10000003854805200.00	1E+16	0.000%
1000000000	1000000042866560000.00	1E+18	0.000%

It certainly appears that as n increases, the difference between f(n) and g(n) decreases into insignificance.

♦ ♦ ♦

Could this be a fluke? Let's try a couple more.

Example 2

$f(n) = n^{1.1} + 3n + 2\lg(n),\ g(n) = n^{1.1}$

n	f(n)	g(n)	% difference
1.E+00	4.000E+00	1.000E+00	-75.000%
1.E+01	4.923E+01	1.259E+01	-74.429%
1.E+02	4.718E+02	1.585E+02	-66.406%
1.E+03	5.015E+03	1.995E+03	-60.216%
1.E+04	5.515E+04	2.512E+04	-54.450%
1.E+05	6.163E+05	3.162E+05	-48.686%
1.E+06	6.981E+06	3.981E+06	-42.974%
1.E+07	8.012E+07	5.012E+07	-37.444%
1.E+08	9.310E+08	6.310E+08	-32.225%
1.E+09	1.094E+10	7.943E+09	-27.414%
1.E+10	1.300E+11	1.000E+11	-23.077%
1.E+11	1.559E+12	1.259E+12	-19.244%
1.E+12	1.885E+13	1.585E+13	-15.916%
1.E+13	2.295E+14	1.995E+14	-13.070%
1.E+14	2.812E+15	2.512E+15	-10.669%

[3] Remember that lg(n) is the log base 2 of n, ie $\log_2(n)$.

[4] Note that this is the percentage change from *f*(*n*) to *g*(*n*), which is lower, hence the percentage change must be negative.

1.E+15	3.462E+16	3.162E+16	-8.665%
1.E+16	4.281E+17	3.981E+17	-7.008%
1.E+17	5.312E+18	5.012E+18	-5.648%
1.E+18	6.610E+19	6.310E+19	-4.539%
1.E+19	8.243E+20	7.943E+20	-3.639%

Once again, the difference appears to be heading towards 0, at least in percentage terms. Note that, as the highest power of n is only 1.1 here compared with 2, it takes longer for the faster term to completely dominate the slower terms.

♦ ♦ ♦

Example 3

$f(n) = 2^n + n^2,\ g(n) = 2^n$

n	f(n)	g(n)	% difference
1	3	2	-33.333333%
2	8	4	-50.000000%
4	32	16	-50.000000%
8	320	256	-20.000000%
16	65792	65536	-0.389105%
32	4294968320	4294967296	-0.000024%
64	1.84467E+19	1.84467E+19	0.000000%
128	3.40282E+38	3.40282E+38	0.000000%
256	1.15792E+77	1.15792E+77	0.000000%
512	1.3408E+154	1.3408E+154	0.000000%

Note the jump in % difference when n=2. This is because n^2 jumps from 1 to 4. However the % difference soon drops off. Note the fastest term here is 2^n, which is a very fast growing function.

Note that the faster the dominant term grows, the quicker it overwhelms the others.

♦ ♦ ♦

This leads us to the principle that to find the theta notation of a function, we should look for the fastest growing term. This will give us the theta notation, which admittedly then must be proved. But how do we know which terms grow faster than others? Well, this is fairly well known already from function theory. No new maths is required. Follow the basic principles below.

1. Powers of n are faster than lg(n).
2. The higher the power of n, the faster the term. So n^3 is faster than n^2, etc.
3. Exponentials are faster than powers, so 2^n is faster than n^{20}, for example.
4. Factorials are faster than exponentials, so $n!$ is faster than 2^n or 3^n.

Try choosing the faster term from the following. Cover up the answers below first!

1. $n^{2.4}, n^{5.8}$
2. $\sqrt{n}, n^2$
3. 2^n, $\lg(n)$
4. n^{120}, $n!$
5. $n * \lg(n)$, n
6. $\lg(n), \sqrt{n}$

<u>Answers</u>

1. $n^{5.8}$
2. n^2 $(\sqrt{n} = n^{0.5})$
3. 2^n
4. $n!$
5. $n * \lg(n)$
6. $\sqrt{n}$

Under construction! More notes to follow soon! In the meantime, I hope this has helped. Cheers, Garry.